Analysis Report on Inspecting MQTT

Liyao Tang - u6142160

May 22, 2018

1 Analysis on Handshakes under Different QoS

Figure 1 shows the required screenshots. QoS defines the handshake of the sending and receiving of one message between the sender and receiver, which both can be either broker or devices. For each QoS level, an explanation of its handshake are given as follow.

1. QoS = 0

   The application message is delivered according to the best efforts of the underlying TCP/IP network and sender would discard the application message once sent out.

   Hence, the application message will arrive at receiver at most once.

2. QoS = 1

   After application message sent, sender waits for an acknowledgement (PUBACK or Publish Ack) to make sure the application message is received. To match PUBACK with corresponding application message, each application message at this QoS level has an ID. After PUBACK received, application message can be safely discarded.

   After a predefined time without returning PUBACK for the application message, sender will re-send application message with its DUP flag set, meaning this is duplicate.

   Hence, the application message will arrive at receiver at least once.

3. QoS = 2

   After application message sent, sender waits for an acknowledgement (PUBREC or Publish Received) to ensure application message is received and then responds with a further acknowledgement (PUBREL or Publish Release) to acknowledge that it knows application message is received. Afterwards, sender still needs to wait for one more acknowledgement (PUBCOMP or Publish Complete) from receiver so that sender is sure that receiver has received the PUBREL. Finally, the application message can be safely discarded.

   After a predefined time without the expecting message, the protocol (either sender or receiver) will retry from the last unacknowledged message.

   Hence, the application message will arrive at receiver exactly once.

No.	Time	Source	Destination	Protoc	Length	Info
4	0.077017…	150.203.213.195	52.65.194.50	MQTT	118	Connect Command
6	0.158035…	52.65.194.50	150.203.213.195	MQTT	72	Connect Ack
8	0.158642…	150.203.213.195	52.65.194.50	MQTT	90	Subscribe Request
10	0.324578…	52.65.194.50	150.203.213.195	MQTT	73	Subscribe Ack
12	0.405582…	52.65.194.50	150.203.213.195	MQTT	92	Publish Message
14	0.982773…	52.65.194.50	150.203.213.195	MQTT	90	Publish Message
16	1.966636…	52.65.194.50	150.203.213.195	MQTT	90	Publish Message
18	1.967216…	150.203.213.195	52.65.194.50	MQTT	70	Disconnect Req

(a) QoS = 0

No.	Time	Source	Destination	Protoc	Length	Info
4	0.035038…	150.203.213.195	52.65.194.50	MQTT	118	Connect Command
6	0.116371…	52.65.194.50	150.203.213.195	MQTT	72	Connect Ack
8	0.117783…	150.203.213.195	52.65.194.50	MQTT	90	Subscribe Request
9	0.197431…	52.65.194.50	150.203.213.195	MQTT	73	Subscribe Ack
11	0.278441…	52.65.194.50	150.203.213.195	MQTT	94	Publish Message
13	0.279013…	150.203.213.195	52.65.194.50	MQTT	72	Publish Ack
15	0.855686…	52.65.194.50	150.203.213.195	MQTT	92	Publish Message
16	0.855944…	150.203.213.195	52.65.194.50	MQTT	72	Publish Ack
18	1.918678…	52.65.194.50	150.203.213.195	MQTT	92	Publish Message
19	1.919294…	150.203.213.195	52.65.194.50	MQTT	72	Publish Ack
20	1.919525…	150.203.213.195	52.65.194.50	MQTT	70	Disconnect Req

(b) QoS = 1

No.	Time	Source	Destination	Protoc	Length	Info
4	0.012693…	150.203.213.195	52.65.194.50	MQTT	118	Connect Command
6	0.093942…	52.65.194.50	150.203.213.195	MQTT	72	Connect Ack
8	0.094997…	150.203.213.195	52.65.194.50	MQTT	90	Subscribe Request
9	0.175283…	52.65.194.50	150.203.213.195	MQTT	73	Subscribe Ack
11	0.260731…	52.65.194.50	150.203.213.195	MQTT	94	Publish Message
13	0.261244…	150.203.213.195	52.65.194.50	MQTT	72	Publish Received
14	0.341834…	52.65.194.50	150.203.213.195	MQTT	72	Publish Release
15	0.342358…	150.203.213.195	52.65.194.50	MQTT	72	Publish Complete
17	0.996354…	52.65.194.50	150.203.213.195	MQTT	92	Publish Message
18	0.996784…	150.203.213.195	52.65.194.50	MQTT	72	Publish Received
20	1.076962…	52.65.194.50	150.203.213.195	MQTT	72	Publish Release
21	1.077539…	150.203.213.195	52.65.194.50	MQTT	72	Publish Complete
23	2.064348…	52.65.194.50	150.203.213.195	MQTT	92	Publish Message
24	2.064815…	150.203.213.195	52.65.194.50	MQTT	72	Publish Received
26	2.145230…	52.65.194.50	150.203.213.195	MQTT	72	Publish Release
27	2.145740…	150.203.213.195	52.65.194.50	MQTT	70	Disconnect Req

(c) QoS = 2

Figure 1: Figures of MQTT handshake under different QoS. The client disconnects the broker after receives three messages.

As listed above, those assisting messages that is not compulsory for application to meet its specification can transmit on QoS level 0; whereas those key messages, which may results in deviation from the specification if not received, should transmit under QoS level 1; while some important messages whose not only payload but also occurrence matter should transmit under QoS level 2 because duplicate messages are not allowed under this circumstance.

2 Statistical Analysis for Each QoS Level

2.1 Collecting Statistics

2.1.1 Code Structure and Explanation

The code is written in python3.6.4, using MQTT library and several other standard python libraries, including OptionParser, time, sys and gc. Especially, MQTT library should be under the same directory of the python script, as configured in my submitted package.

The code takes two compulsory command line parameters, explained as follow:

- "- -speed" It takes one of following values: ("fast", "slow", "SYS"), where "fast" stands for the fast channel, "slow" for the slow channel and "SYS" for the "$SYS/broker/#" channel.
- "- -qos" It takes one of following values: (0, 1, 2), which corresponds to the MQTT qos level of. If the "–speed" takes "SYS", then "–qos" must be 2.

If the script subscribes to one of the fast channels, it will collect the statics for 1 minutes and record it into a local log and will publish the required statics after 10 minutes. Then it will starts another round of collecting statics, recording logs and publishing result. All the recorded time are local time-stamp.

If the script subscribes to the "$SYS/broker" topic, it will write whatever it receives into a local log file with a local time-stamp.

In practice, four scripts will run in parallel, outputting into separate files, referring to the local time on the same computer when time-stamping messages. It is shown that the script is able to run for days with a nearly constant memory consumption on a PC.

2.1.2 Examples to Run the Code

- Run the python script:
  - to subscribe to topic: counter/fast/q0:

```
python ./Inspector.py --speed fast --qos 0
```

- to collect statics from $SYS broker #:

  ```
  python ./Inspector.py --speed SYS --qos 2
  ```

- Run the python script in parallel:

  I provide a simple bash script, which needs to be placed under the same directory with the python script "Inspector.py" and requires a "./Log" directory in its directory. The script will subscribe to topics "counter/fast/q0", "counter/fast/q1", "counter/fast/q2" and "$SYS broker #"; and output the log into corresponding files under "./Log", with a time stamp of its starting time as postfix of each log.

  To run the script in background:

  ```
  bash scan_channel.sh &
  ```

2.1.3 Statics Definition

As we are requested multiple statics, some special name will be used in the following report are first introduced and then it will be given the concrete definition and calculation for those statics and some other statics collected in my code for the purpose of analysis.

Special name:

- valid message If the client subscribes to one of the fast channel then a valid message is any message from the topic whose payload can be directly translated into an integer.

- expected number If the client subscribes to one of the fast channel and the latest number in received valid messages is x then either $x+1$ ot 0 is the expected number. Yet, if the client subsribes to one of the fast channel and has not received any number from a valid message from the topic, than any integer is expected.

- ordered message If the client subscribes to one of the fast channel and the latest time stamp in received valid messages is t, than ordered message is a valid message with a time stamp t' such that $t' > t$. Yet, if the client subsribes to one of the fast channel and has not received any valid message from the topic, than any valid message is ordered.

- expected message If the client subscribes to one of the fast channel then the expected message is a ordered message containing a expected number.

- duplicate message The duplicate messages are considered only under qos 1 and 2 because the broker by definition should not re-send any message under qos level 0.

  Under qos level 1 or 2, if the client subscribes to one of the fast channel, duplicate message is any ordered message with the same number as the current one. (An ordered message with a number bigger or smaller than the current one are considered an loss in messages that are, correspondingly, either before or after wraps around.)

- mis-ordered message

  If the client subscribes to one of the fast channel then any message that is not considered ordered message is a mis-ordered message.

Definition of statics:

- 1-min expected count

  The expected count are the total number of expected message within the 1-minute interval.

- 1-min rate of messages received

  This statics are the number of valid messages received in the 1-minute interval.

- 1-minute loss rate

  This statics are calculated by following equation 1.

$$\text{loss rate} = \frac{losscount}{count} \tag{1}$$

  where *count* is the 1-min rate of messages received and *losscount* are the lost messages within this 1-minute interval. The messages loss are recognised whenever it is received an ordered message with a number not consistent with the expected number. The worst 1-minute loss rate is the biggest loss rate encountered in 10-minute interval.

- worst 1-minute duplicate rate

  Similar to 1-minute loss rate, it is calculated by equation 2

$$\text{loss rate} = \frac{dupecount}{count} \tag{2}$$

  where *count* is the 1-min rate of messages received and *dupecount* are the duplicate messages with in this 1-minute interval. Worst 1-minute duplicate rate is then the biggest 1-minute loss rate encountered in the 10-minute interval.

- 1-minute out-of-order rate

  out-of-order rate is calculated by equation 3

$$out - of - orderrate = \frac{mis - ordercount}{count} \tag{3}$$

  where *count* is the 1-min rate of messages received and $mis-ordercount$ is the number of mis-ordered message encountered in the 1-minute interval. Worst 1-minute out-of-order rate is then the biggest 1-minute out-of-order rate encountered in the 10-minute interval.

2.2 Correlation with $SYS

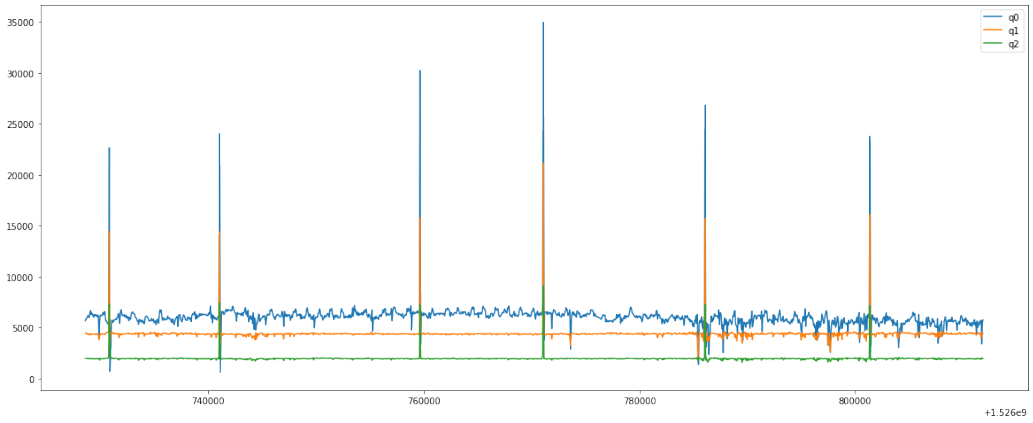
Besides the required fields, there are other topics analysed in the following analysis and they are labelled correspondingly in their graphs. The following graph are the time span from 2018-05-19 21:17:50 to 2018-05-20 20:22:14.

Convention of plotting in this section is that the information collected by clients are plotted in either blue, orange or green while the SYS statics are in red. The related discussions are placed under each graph and cross-referenced.

2.2.1 1-min Expected Messages Correlating with $SYS

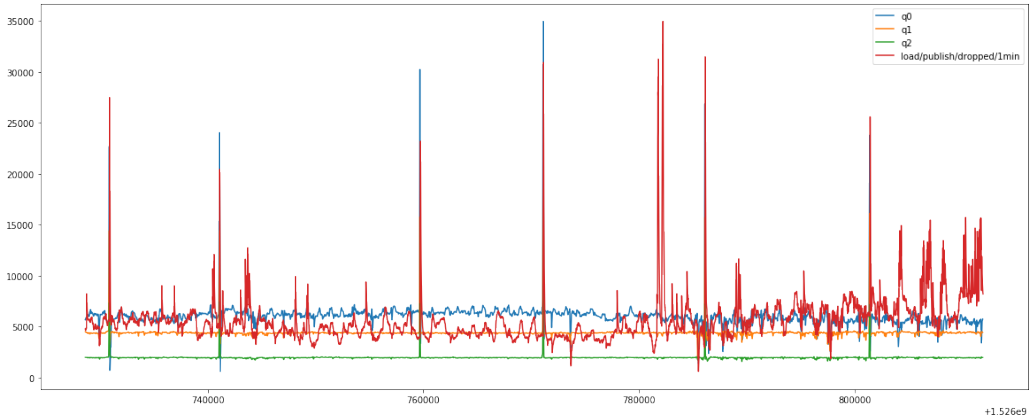
To measure the performance of the broker along the time series, the number of expected messages over 1 minute under each QoS level are plotted into graphs. In order to show the correlation with SYS statics, the SYS statics are re-scaled appropriately so that the trends and details of both number of expected messages and the SYS statics can be expressed.

Figure 2: 1-min Expected Messages under QoS 0,1,2



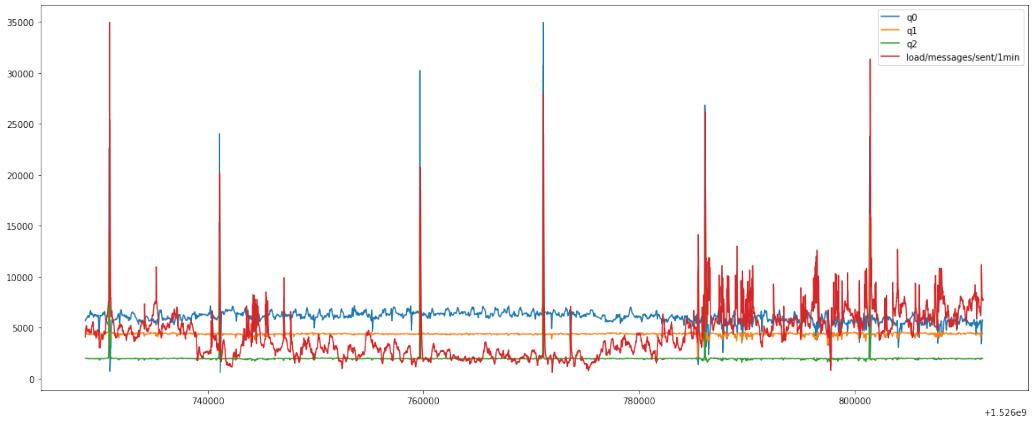
As convention, the blue, orange and green curves are 1-min expected messages number under QoS level respectively 0, 1 and 2.

Figure 3: topic: load/publish/dropped/1min



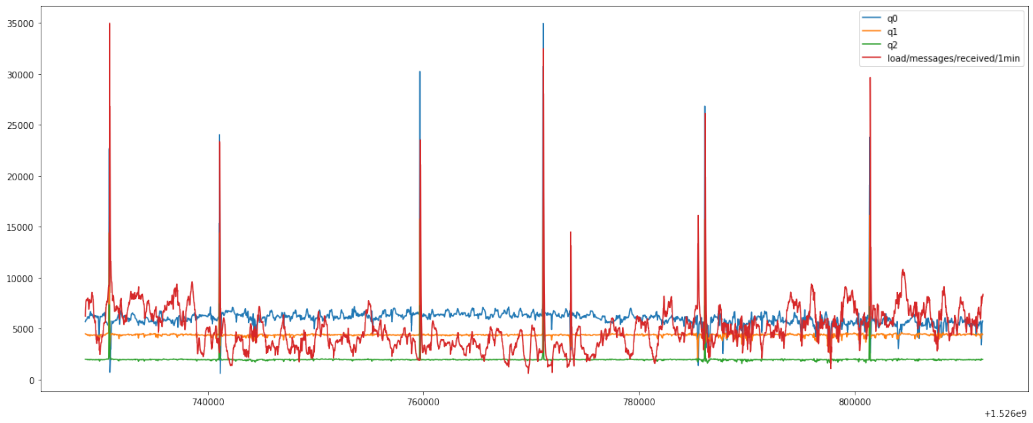
It is observed that, surprisingly, the dropped messages largely correlates with the number of received expected messages, except for the fifth and jitters in the latter part (after the 3rd x-tick); it is reasonable because each peak is actually a publishing peak (in figure 4), when messages is dropped due to the jam. It is also noticed that curves for expected received messages has more jitters in the latter part as well, correlating with the dropped/1min curve.

Figure 4: topic: load/messages/sent/1min



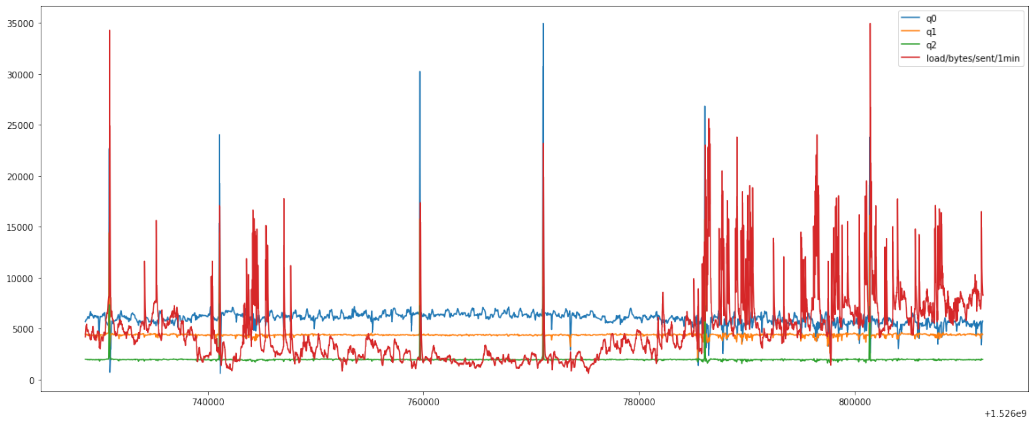
All the peak of 1-min expected messages number for each QoS nicely correlates with the sending peak from the SYS statics, which is expected because more it sends, more clients receive, and more chance for client to have an expected sequential stream of received number.

Figure 5: topic: load/messages/received/1min



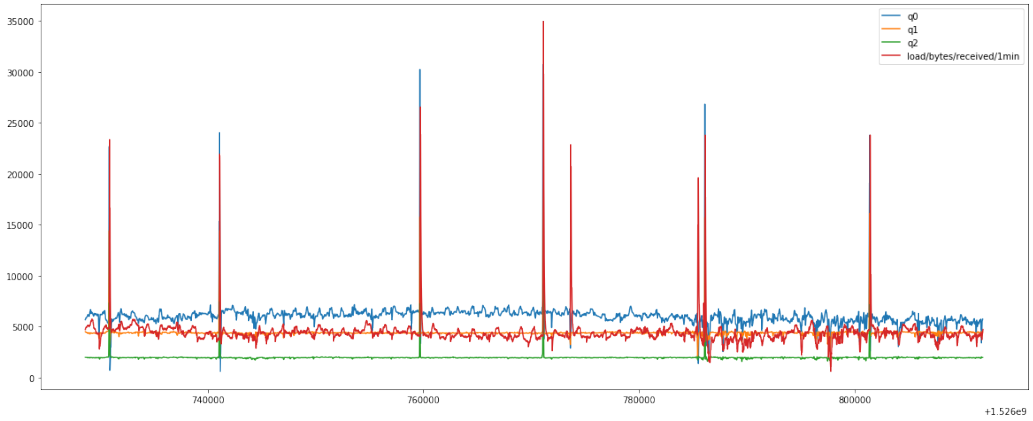
Again, all the peaks from SYS statics, except for some small peaks, correlates with curves for expected messages. This is expected because messages received by clients fundamentally come from the fast counter and broker needs to receives those numbers before passing them to clients.

Figure 6: topic: load/bytes/sent/1min



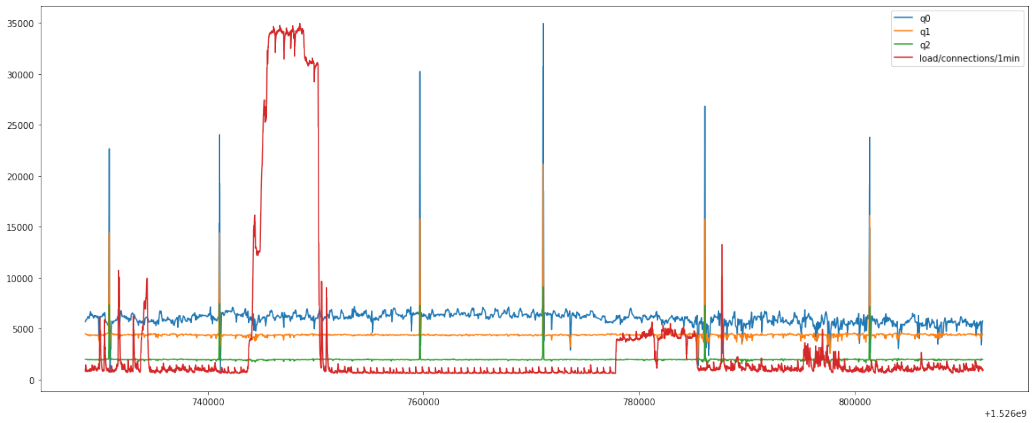
This is actually an interesting plot because it accounts for the size of each messages, where we can observe that on the left hand side around the first x-tick (740000) and after the third x-tick (780000), there are a number of small peaks of sending, denoting peaks of broker's work load for sending messages, which nicely correlates with those jitter-like peaks in previous figures (load-/message/sent/1min: fig 4, load/publish/dropped/1min: fig 3)

Figure 7: topic: load/bytes/received/1min



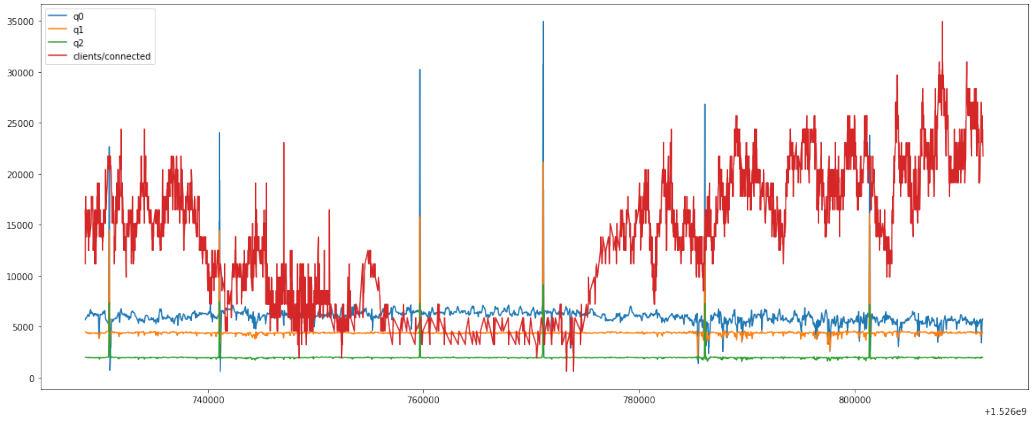
It is however interesting that the messages received by the broker is not consistent with those outgoing peaks, especially those jitters. Then why would broker suddenly needs to send bigger and more messages?

Figure 8: topic: load/connections/1min



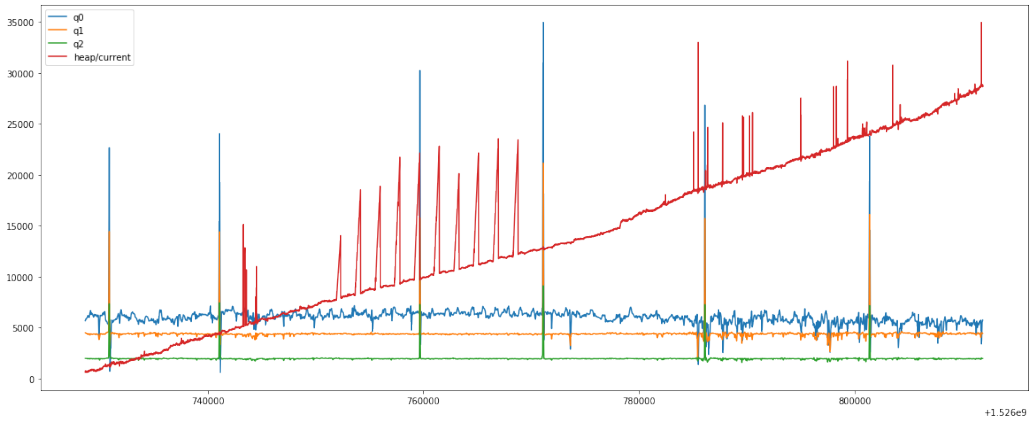
There is a burst of client requesting to connect but it seems to not correlate those jitters, indicating CONNECT packets itself would not have any instant impact.

Figure 9: topic: clients/connected



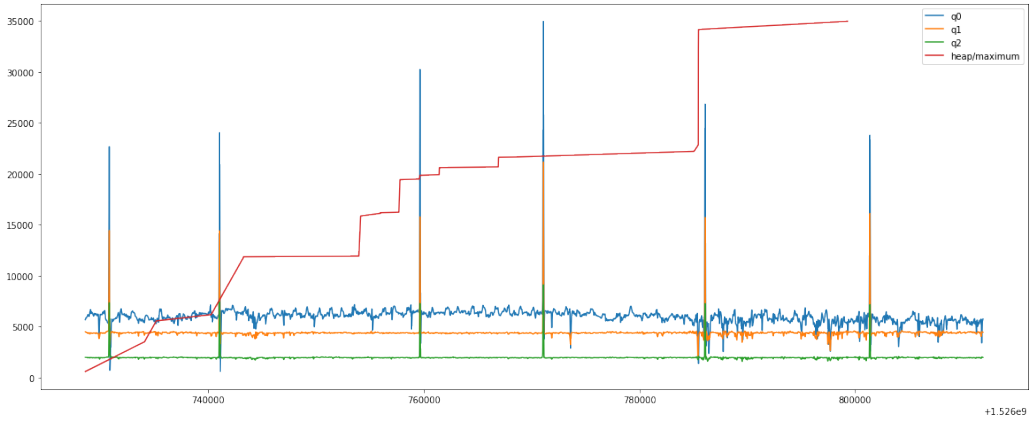
It is clear now in this figure that the number of active clients coorelates with those small peaks, or previously referred as jitters, where broker needs to send more messages and bytes out; and it is reasonable that broker sends more at when more clients connect and receive messages from broker, especially they may receive messages from other topic then topics subscribed by my clients, i.e. slow channel.

Figure 10: topic: heap/current



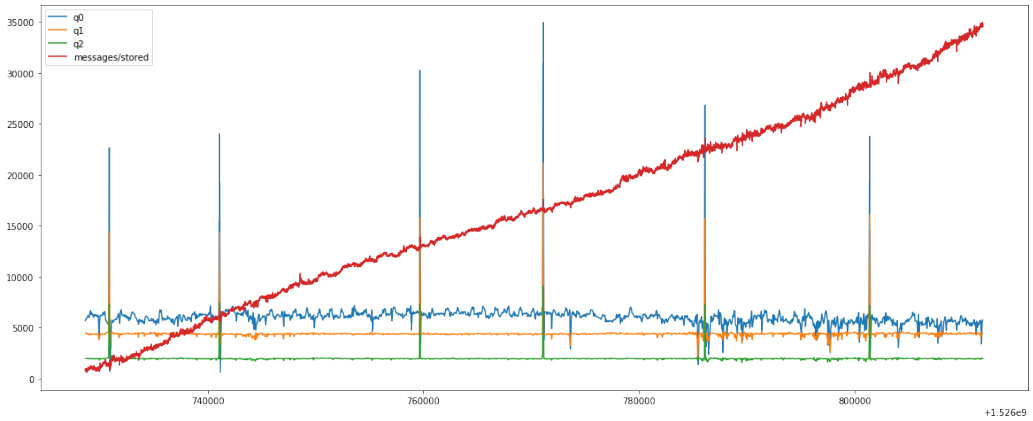
It is expected to observe an increasing allocated memory on the broker because intuitively there will be more and more retained messages published from students to the broker; it actually nicely correlates with the number of retained messages in figure 13 as expected. It is yet unexpected to see those small peaks along its increasing, denoting several memory burst and I could not find any coorelated curves with that in the given time of this assignment. It also seems to not correlate with the number of received expected messages.

Figure 11: topic: heap/maximum



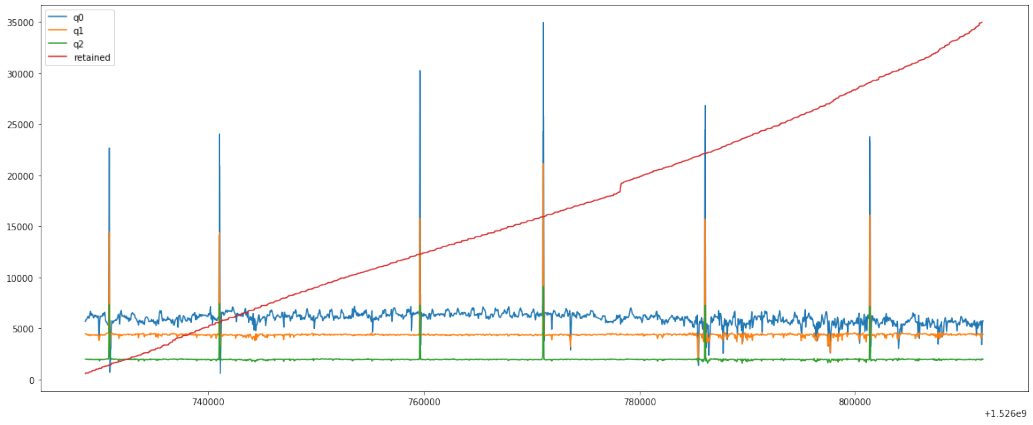
The maximum heap memory ever allocated is expected and correlates with the trends of the curve for current heap memory.

Figure 12: topic: messages/stored



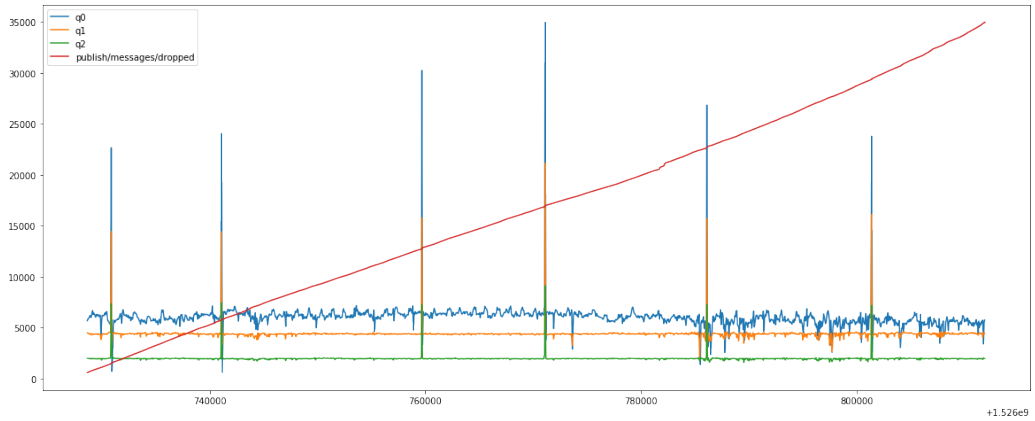
This is messages held in message store, including retained messages and in-queue messages waiting for delivery and being acknowledged under QoS level 1 and 2, which, as expected, correlates with retained messages and current heap memory, and seems to not correlate with the number of received expected messages.

Figure 13: topic: retained



The number of retained messages seems to be held in the heap memory and not correlates with or affects the expected messages received by clients or

Figure 14: topic: publish/messages/dropped



This is the total number of dropped messages due to inflight/queuing limits since the start, which is unexpected to not correlates with the number of clients' received expected messages, the received messages on the broker or the current heap memory allocated on the broker. The potential reason might be that the queues are allocated with a specialised memory instead of on the heap memory.

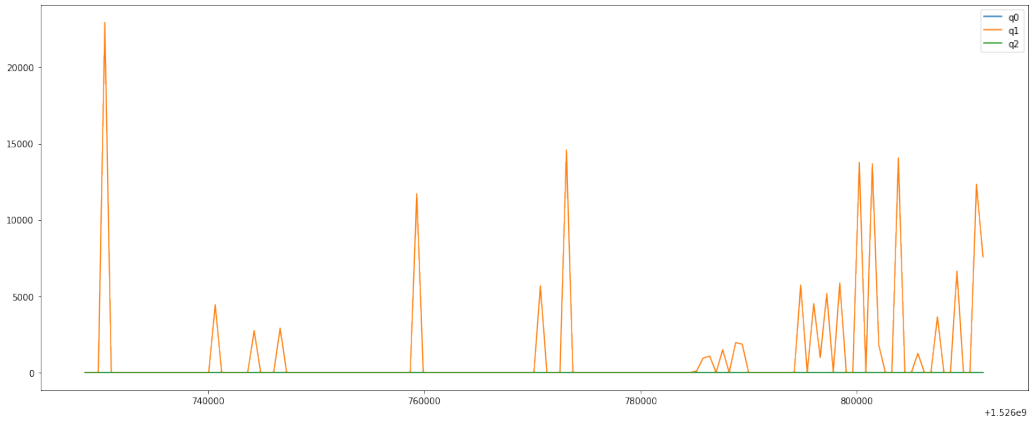
2.2.2 10-min Loss Rate Correlating with $SYS

In this section, instead of using the total received expected message over 1 minute, we analyse the broker performance with loss rate, calculated with equation 1. The out-of-order rate or duplicate rate are not analysed because they are too rare and thus do not have sufficient data.

It is surprising that during the period of this measurement, QoS level 0 is found to have 0 loss and QoS level 2 has a loss rate close to 0, compared to QoS level 1. Hence, the following graph will contain only curves from QoS level 1 and 2 on the background.

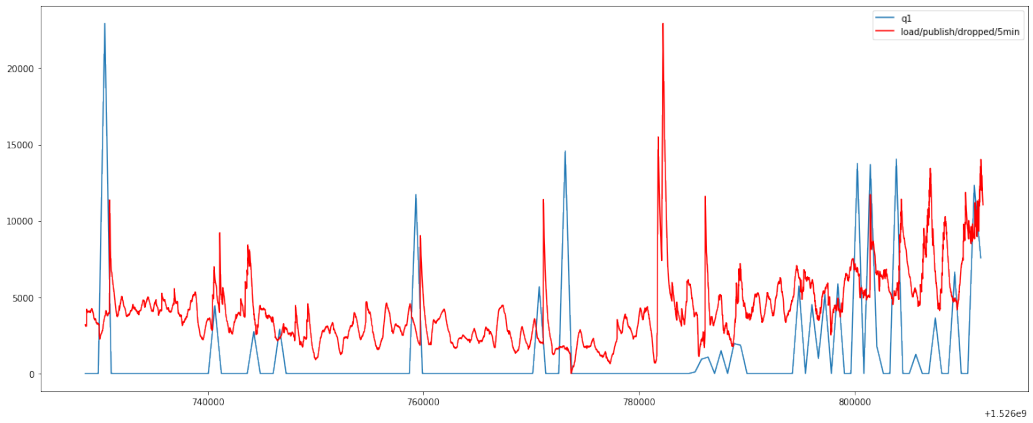
Similar to that in previous section 2.2.1, the discussions are under each corresponding figure.

Figure 15: 10-min loss rate under QoS 0,1,2



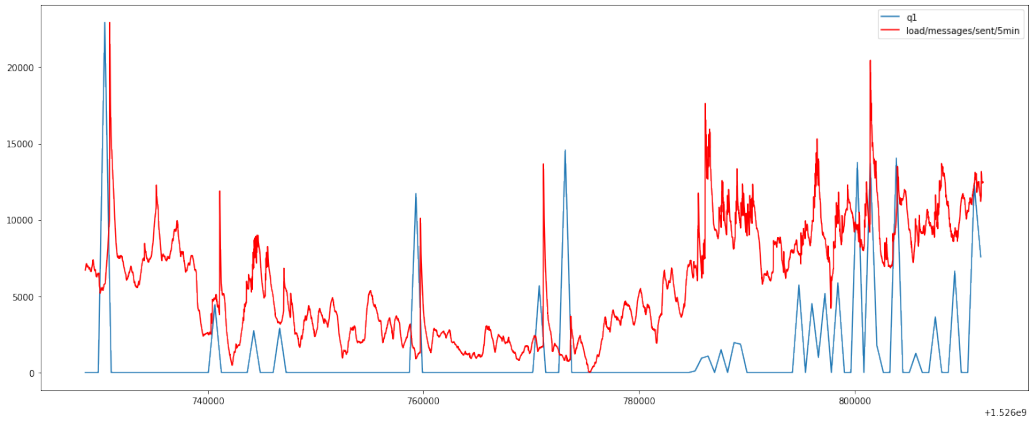
As convention, the blue, orange and green curves are 10-min expected messages number under QoS level respectively 0, 1 and 2. The loss rate of QoS level 0 is 0 and level 2 close to 0. Thus we only show the loss rate curve of QoS level 1 when comparing with SYS statics.

Figure 16: topic: load/publish/dropped/5min



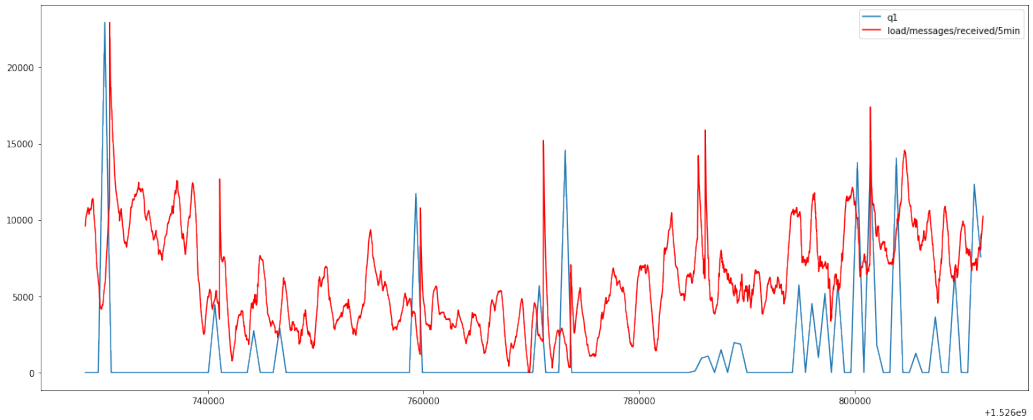
It is observed that the dropped messages correlates with the loss rate as expected. It is interesting that the server becomes more and more stressed as the time goes, uncovered by the density of peaks in loss rate, which is denser in the latter part.

Figure 17: topic: load/messages/sent/5min



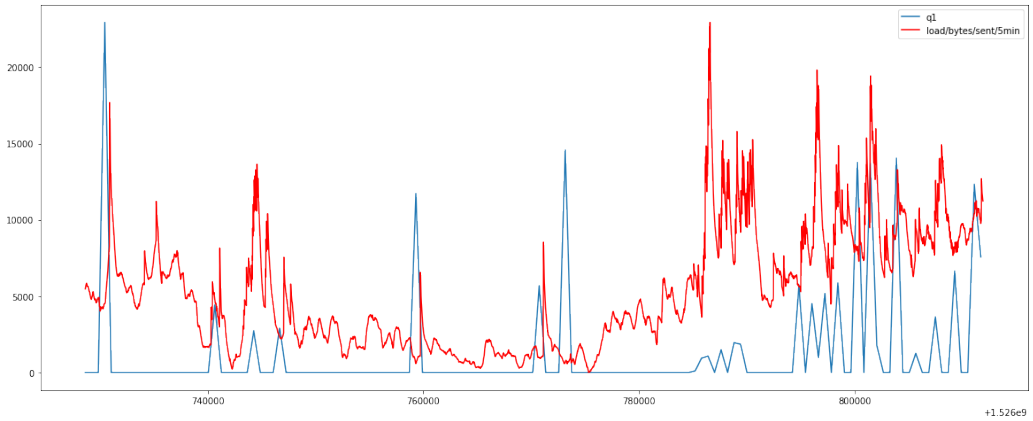
The loss rate curve barely correlates with the sending messages statics from broker. When broker needs to send more messages at latter part, the loss rate curve has more peaks. Or, in the other word, the more it needs to send, the more likely clients have lost messages.

Figure 18: topic: load/messages/received/5min



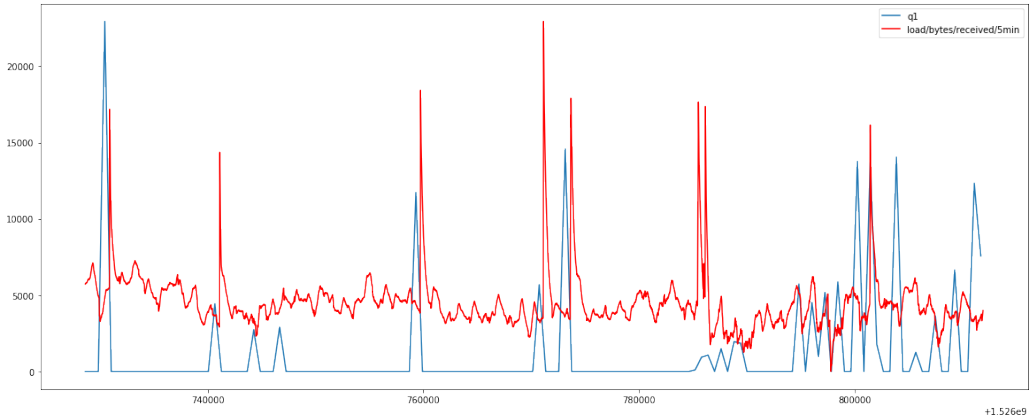
The curve for received messages at broker seems not to correlate with the loss rate curve, which is expected because the received messages are largely from counter on a relatively steady pace.

Figure 19: topic: load/bytes/sent/5min



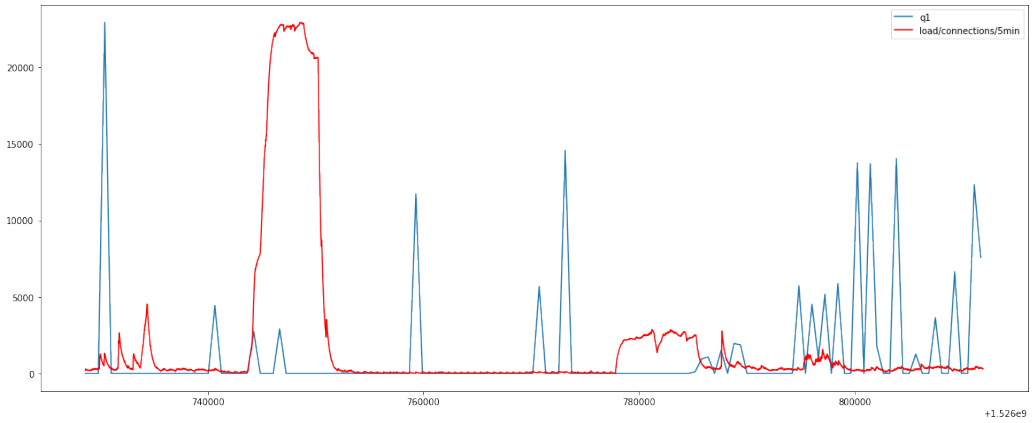
Apart from the correlation within SYS statics, (discussed in figure 6), it also largely correlate with the loss rate curve, which is reasonable as more burden on the broker's task of sending will result in more loss rate.

Figure 20: topic: load/bytes/received/5min



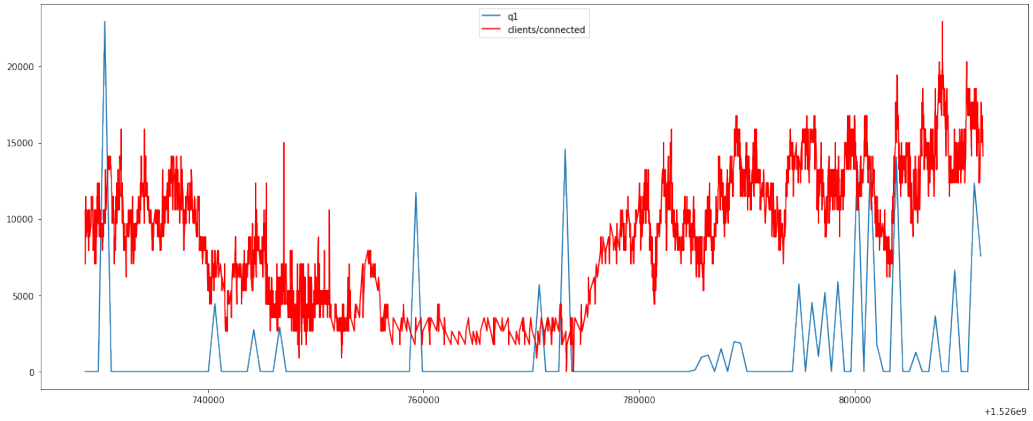
This figure becomes interesting because curve for received bytes starts from largely correlating with loss rate curve and then ends up with not quite correlated. Such phenomenon is understandable because the received bytes over 5 minutes should reveal the burden of the broker's task of handling messages and denominates the loss rate, until another factor, the number of client, becomes large enough to affect the loss rate in the latter of this record.

Figure 21: topic: load/connections/5min



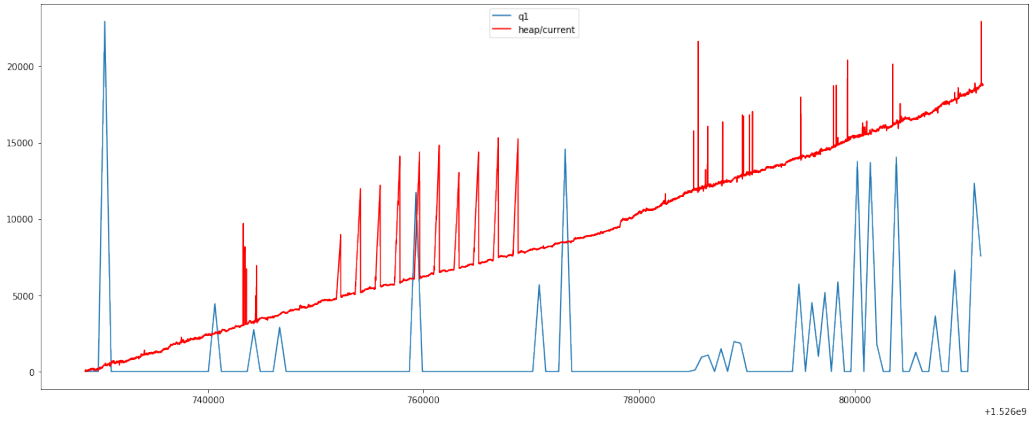
Similar to previous section, the CONNECT packets do not directly impact or correlate with the clients' statics, loss rate in this case.

Figure 22: topic: clients/connected



As discussed in previous section 2.2.1, the active clients are a key role in influencing broker's performance. We observe and expect to observe that as more clients connected to the broker in the latter part, more peaks in loss rate curve show up.

Figure 23: topic: heap/current



As discussed in previous section 2.2.1, the current heap memory correlates with the number of both retained and stored messages in broker and also the maximum heap memory used. Yet it is not considered correlated with clients' statics, loss rate in this case. Hence, for simplicity, the analysis for correlation between loss rate curve and curves for topics of heap/maximum, messages/stored and retained are not shown here; and the result is the same as the analysis between curves for expected messages and those topics.

3 Analysis and Correlation between Students' Report

4 Real-World Analysis

Consider the broader end-to-end network environment, in a situation with many thousands of sensors publishing regularly to some large number of subscribers. Explain in your report

4.1 Performance Challenges

4.1.1 End-network: Sources and Clients

The challenge at source would be dependent on actual task. In the case of master-slave model, where the source are only responsible for publishing messages and performs only one-way communication, the challenge for CPU and memory of source are small, because it only needs to support its own use. It is yet challenging for sources to dynamically recognise and adapt to the clients requirement or constraint. Such challenge will transit to the clients, whose CPU and memory should be able to catch up with the sources and cope with potential incoming messages burst.

While in the case where two-way communication between clients and sources exists, the challenge of adapting to each other are reduced; nonetheless the sources need to cope with different requirement from various clients, which can forms a challenge for the sources' ability of offering the services in different forms or under different conditions in parallel.

The network challenge here is the well-known last-mile challenge. The last-mile network can be under different situation and exposed to various noise, which potentially requires the sources and clients to be aware of such condition if used in the IoT / IoE architecture.

4.2 Intermediate: Broker

The broker in MQTT becomes a bottleneck, also known as single failure point, of the overall performance. It is because all the messages need to first collected into the broker and then distributed to the corresponding clients in their preferred way.

As also mentioned in challenges for sources, the broker needs to have enough ability to provide three different services. It also becomes a huge challenge for CPU and memory on the broker because even if a few sources happen to have a output burst it will add up and potentially arrives together on the broker.

The memory consumption is another challenge because not only messages on the fly needs enough space for itself or acknowledgement, but also retained messages are likely to constantly take up spaces on the broker and might cause the service to degrade when free memory is low.

The network at the broker is crucial, because the network is not only more likely to be jammed because of output burst from the sources, but also can heavily influence the efficiency of broker's sending out the messages in queue. If the outgoing network is always not as efficient as the incoming network, the broker's limited memory is going to run out eventually.

4.3 Different QoS Levels for Challenges

To ease the pain of sources not knowing the performance limit on clients, QoS level 1 or 2 can be helpful, depending on if the occurrence of message matters, as discussed under figure 1. Because in QoS level 1 or 2, broker handles the acknowledgement of the messages on behalf of the sender.

However, QoS level 1 or 2 can brings server burden onto the broker, because messages need to take up the resources on broker both longer and more. Such burden may add up to a point where broker needs to break the specification in protocol due to physical limits.

To prevent the bottleneck from limiting the overall performance, it is suggested to use QoS level 0, where sources and clients may need to handle the acknowledgement and adaptation by themselves. In return, the broker is able to have less CPU and memory burden. Nonetheless, it is worth noting that the network burden may not be reduced or may even be enhanced if clients and sources decide to handle acknowledgement by themselves. Overall, QoS level 0 can reduce the broker's stress.

4.4 Comparison with Measurement

Interestingly, regarding averaged 10-min loss rate, the QoS level 0 performs the best, with 0.0 10-min loss rate; the QoS level 2 the second, with 1.4 10-min loss rate; and the QoS level 1 the worst, with 5259 10-min loss rate.

While regarding the number of received expected messages, QoS level 0 , 1 and 2 reaches an average of 5842, 4620 and 2203 over 1-min interval during the measurement. The measurement here are all from 2018-05-19 21:17:50 to 2018-05-20 20:22:14, measured in parallel.

It is expected that QoS level 0 actually performs the best because that part of broker is able to cater everybody on a nearly constant pace as the incoming speed is almost constant and the number of clients are roughly 500 according to the SYS statics. Given a relatively stable network environment in ANU (thanks to network team), the performance is close to perfect as expected. Hence, QoS level 0 might be able to send at a comparable speed with the speed for incoming messages from counter; and thus achieving a 0.0 loss rate and the most messages received on average.

It is yet unexpected that the QoS level 2 would outperforms the level 1, regarding the loss rate. The proposed reason is that the sources publish message in QoS level 1 and may jam the corresponding memory more often because it does not care the duplication. In other word, QoS level 2 is able to stop for a little while because of its 4-way handshake, whereas in QoS level 1 there is no gap between the publish of two numbers, causing the broker to be under a relatively high burden in terms of available slots in messages queue.

The above proposed reason is also consistent with the total expected messages each QoS level receives. That is, the QoS level 1 receives much more than that of QoS level 2 in 1-min interval and thus operating in high speed and stressing the broker out more often and QoS level 2 receives on average only a half of the QoS level 1 over the 1-min interval.